(4)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-492
December 1988

Universal Packet Routing Algorithms

Tom Leighton, Bruce Maggs, and Satish Rao

Abstract

In this paper we examine the packet routing problem in a network independent context. Our goal is to devise a strategy for routing that works well for a wide variety of networks. To achieve this goal, we partition the routing problem into two stages: a path selection stage and a scheduling stage.

In the first stage we find paths for the packets with small maximum distance, d, and small maximum congestion, c. Once the paths are fixed, both are lower bounds on the time required to deliver the packets. In the second stage we find a schedule for the movement of each packet along its path so that no two packets traverse the same edge at the same time, and so that the total time and maximum queue size required to route all of the packets to their destinations are minimized. For many graphs, the first stage is easy - we simply use randomized intermediate destinations as suggested by Valiant. The second stage is more challenging, however, and is the focus of this paper. Our results include:

1. a proof that there is a schedule of length $O(c+d)$ requiring only constant size queues for any set of paths with distance d and congestion c,

2. a Randomized on-line algorithm for routing any set of N "leveled" paths on a bounded-degree network in $O(c+d+\log N)$ steps using constant size queues,

3. the first on-line algorithm for routing N-packets in the N-node shuffle-exchange graph in $O(\log N)$ steps using constant size queues, and

4. the first constructions of area and volume-universal networks requiring only $O(\log N)$ slow-down.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

Acknowledgements

This research was supported in part by the Defense Advanced Research Projects Agency under Contrack N00014-87-K-0825, the Office of Naval Research under Contract N00014-86-K-0593, the Air Force under Contract OSR-86-0076, and the Army under Contract DAAL-03-86-K-0171. Tom Leighton is supported in part by an NSF Presidential Young Investigator Award with matching funds provided by AT&T Bell Laboratories and IBM. Bruce Maggs is supported in part by an NSF Graduate Fellowship.

Author Information

Leighton: Laboratory for Computer Science and the Department of Mathematics, MIT, Room NE43-836A, Cambridge, MA 02139. (617) 253-5876.

Maggs: Laboratory for Computer Science, MIT, Room NE43-313, Cambridge, MA 02139. (617) 253-7843.

Rao: Laboratory for Computer Science and the Department of Mathematics, MIT, Room NE43-342, Cambridge, MA 02139. (617) 253-5889.

Universal Packet Routing Algorithms

(Extended Abstract)

Tom Leighton[1,2]
Bruce Maggs[2]
Satish Rao[2]

[1] Mathematics Department and
[2] Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

In this paper we examine the packet routing problem in a network independent context. Our goal is to devise a strategy for routing that works well for a wide variety of networks. To achieve this goal, we partition the routing problem into two stages: a path selection stage and a scheduling stage. In the first stage we find paths for the packets with small maximum distance, d, and small maximum congestion, c. Once the paths are fixed, both are lower bounds on the time required to deliver the packets. In the second stage we find a schedule for the movement of each packet along its path so that no two packets traverse the same edge at the same time, and so that the total time and maximum queue size required to route all of the packets to their destinations are minimized. For many graphs, the first stage is easy – we simply use randomized intermediate destinations as suggested by Valiant. The second stage is more challenging, however, and is the focus of this paper. Our results include:

1. a proof that there is a schedule of length $O(c+d)$ requiring only constant size queues for any set of paths with distance d and congestion c,
2. a randomized on-line algorithm for routing any set of N "leveled" paths on a bounded-degree network in $O(c+d+\log N)$ steps using constant size queues,
3. the first on-line algorithm for routing N-packets in the N-node shuffle-exchange graph in $O(\log N)$ steps using constant size queues, and
4. the first constructions of area and volume-universal networks requiring only $O(\log N)$ slowdown.

This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-87-K-825, the Office of Naval Research under Contract N00014-86-K-0593, the Air Force under Contract OSR-86-0076, and the Army under Contract DAAL-03-86-K-0171. Tom Leighton is supported in part by an NSF Presidential Young Investigator Award with matching funds provided by AT&T Bell Laboratories and IBM. Bruce Maggs is supported in part by an NSF Graduate Fellowship.

1 Introduction

1.1 Background

The task of designing an efficient packet routing algorithm is central to the design of most large-scale general purpose parallel computers. In fact, even the basic unit of time in some parallel machines is measured in terms of how fast the packet router operates. For example, the speed of an algorithm in the Connection Machine is often measured in terms of *routing cycles* (roughly the time to route a random permutation) or *petit cycles* (the time to perform an atomic step of the routing algorithm). Similarly, the performance of machines like the BBN Butterfly is substantially influenced by the speed and rate of successful delivery of its router.

Packet routing also provides an important bridge between theoretical computer science and applied computer science; it is through packet routing that a real machine such as the Connection Machine is able to simulate an idealized machine such as the CRCW PRAM. More generally, getting the right data to the right place at the right time is an important, interesting, and challenging problem. Not surprisingly, it has also been the subject of a great deal of research.

1.2 Past work

The first major result in packet routing is due to Batcher [3] who devised an elegant and practical algorithm for routing any permutation of N packets on an N-processor shuffle-exchange graph in $\log^2 N$ steps. The result extends to routing many-one problems provided that (as is typically assumed) combining can be used to merge packets that have a common destination.

No better deterministic algorithm was found until Ajtai, Komlos, and Szemeredi [1] solved a classic open problem by constructing an $O(\log N)$-depth sorting network. Leighton [11] then used this $O(N \log N)$-node network to construct a degree 3 N-node network capable of solving any N-packet routing problem in $O(\log N)$ steps. Although this result is optimal up to constant factors, the constant factors are quite large and the algorithm is of no practical use. Hence, the effort to find fast deterministic algorithms has continued.

Thus far, the best small-constant-factor deterministic algorithm is an $O(\log^2 N/\log\log N)$-step algorithm for routing on the butterfly.

There has been comparatively much greater success in the development of efficient randomized packet routing algorithms. The study of randomized algorithms was pioneered by Valiant and Brebner [25] who showed how to route any permutation of N packets in $O(\log N)$ steps on an N-node hypercube with queues of size $O(\log N)$ at each node. Although the algorithm was not always guaranteed to work, it was guaranteed to work with probability at least $1 - 1/N$ for any permutation. This result was improved in a succession of fundamental papers by Aleliunas [2], Upfal [24], Pippenger [17], and Ranade [18]. Aleliunas and Upfal developed the notion of a *delay path* and showed how to route on the shuffle-exchange and butterfly graphs (respectively) in $O(\log N)$ steps with queues of size $O(\log N)$. Pippenger was the first to eliminate the need for large queues, and showed how to route on a variant of the butterfly in $O(\log N)$ steps with queues of size $O(1)$. Ranade showed how combining could be used to extend the Pippenger result to include many-one routing problems, and tremendously simplified the analysis required to prove such a result. As a consequence of Ranade's work, it has finally become possible to simulate a step of an N-processor CRCW PRAM on an N-node butterfly or hypercube in $O(\log N)$ steps using constant size queues on each edge.

Concurrent with the development of these hypercube-related packet routing algorithms has been the development of algorithms for routing in arrays. Kunde [8] showed how to route any permutation of N packets deterministically in $(2+\varepsilon)\sqrt{N}$ steps using queues of size $O(1/\varepsilon)$. Also, Krizanc, Rajasekaran, and Tsantilis [7] showed how to randomly route any permutation in $2\sqrt{N} + O(\log N)$ steps using constant size queues. Most recently, Leighton, Makedon and Tollis discovered a deterministic algorithm for routing any permutation in $2\sqrt{N} - 2$ steps using constant size queues, thus achieving the optimal time bound in the worst case.

1.3 Our approach

One deficiency with the state-of-the-art in packet routing is that aside from Valiant's paradigm of "first routing to a random destination," all of the algorithms and their analyses are very specifically tied to the network on which the routing is to take place, as well as to the requirement that packets are first routed to destinations that are (in some sense) random. For example, the butterfly routing algorithms are all quite different than the array algorithms in the way that queue size is kept constant. Moreover, the butterfly and hypercube algorithms are so specific to those networks that no $O(\log N)$-step constant-queue-size algorithm was known for the closely related shuffle-exchange graph. The lack of a good routing algorithm for the shuffle-exchange graph is one of the reasons that the butterfly is preferred to the shuffle-exchange graph in practice.

In this paper, we take a significant step towards the development of a universal approach to packet routing. Our approach to the problem differs from previous approaches in that we separate the process of selecting packet paths from the process of timing packet movements along the paths. More precisely, given any underlying network, and any selection of paths for the packets, we study the problem of timing the movement of the packets so as to minimize the total time and maximum queue size needed to route all the packets to their correct destinations.

Of course, there must be some correlation between the performance of the algorithm and the selection of the paths. In particular, the maximum distance d traveled by any packet is always a lower bound on the time required to route all packets, as is the congestion c of the paths. (The *congestion* of a collection of packet paths is the largest number of packets that must traverse a single edge during the entire course of the routing.)

Viewed in terms of these parameters, then, a routing problem can be broken into two stages. In Stage 1, we select paths for the packets so as to minimize c and d. In Stage 2, we schedule the movement of the packets so as to minimize the total time and maximum queue size.

For many networks, Stage 1 is easy. We simply use Valiant's paradigm of first routing to a random destination, and then routing to the correct destination. It is easily shown for arrays, butterflies, shuffle-exchange graphs, etc., that this approach yields values of c and d that are within a small constant factor of the diameter of the network, which is as well as can be done. Moreover, this technique also usually works for many-one problems provided that the address space is randomly hashed.

Stage 2 has traditionally been the hard part of routing. Curiously, however, we have found that by ignoring the underlying network and the method of path selection, Stage 2 actually becomes easier to solve! Hence we will be able to obtain results for routing that are both simpler and far more general than existing approaches. Among other things, we will be able to route on the N-node mesh in $O(\sqrt{N})$ steps using constant size queues with the same algorithm that uses $O(\log N)$ steps and constant size queues on the butterfly. We will also be able to route on the shuffle-exchange graph in $O(\log N)$ steps with constant size queues. Also, we provide the first examples of volume and area-universal networks that require only $O(\log N)$ slowdown by showing how to route efficiently on a fat-tree.

1.4 Outline of the results

Our most difficult result is a proof that any set of packet paths with congestion c and distance d can be scheduled so as to complete the routing in $O(c+d)$ steps using constant size queues. This result is optimal up to constant factors, and substantially improves the naive bound of $O(cd)$ steps and $O(c)$ size queues. Unfortunately, the result is highly nonconstructive, and therefore is useful only if substantial amounts of off-line computation are available for the routing. On the other hand, the result is robust in the sense that it provides a near-optimal schedule of packet movements for any set of paths and any underlying network. Such robustness is particularly useful when dealing with routing problems on arbitrary distributed networks as in [12]. The proof of the result is contained in Section 2.

We do not know whether or not there is an on-line algorithm that can route any set of paths in $O(c+d)$ steps with constant size queues. It is not difficult to devise a randomized on-line algorithm to schedule any set of N paths in $O(c+d\log N)$ steps using queues of size $O(\log N)$. In special cases, however, we can do better. For example, a slight variant of Ranade's algorithm can be used to schedule on-line any leveled set of N paths on a bounded-degree network in $O(c+d+\log N)$ steps using constant size queues. By a *leveled* set of paths, we mean a set of paths for which each packet starts from a level one node, progresses from a level i node to a level $i+1$ node at each step, and ends at a level d node. For example, greedy paths on the butterfly are leveled in this fashion. The algorithm is randomized, but requires only $\Theta(\log N \log\log N)$ bits of randomness to succeed with high probability. The proof of this result is included in Section 3. Curiously, the proof is simpler than the previous proof of the same result applied specifically to routing random paths in butterflies [18]. (The fact that Ranade's algorithm can be used in this general context has also been observed by Ranade [19].)

The on-line algorithm for leveled networks can immediately be applied to obtain good routing algorithms for arrays and butterflies. With some extra effort, it can also be applied to obtain the first constant queue size algorithm for routing on the shuffle-exchange graph. It can also be applied to construct a class of networks that are *area universal* in the sense that the network in the class with N processors has area $O(N)$, and can, with high probability, simulate in $O(\log N)$ steps each step of any other network of area $O(N)$. An analogous result is shown for a class of *volume universal* networks. The details of these applications are included in Section 4.

This paper leaves open the question of whether or not there is an on-line algorithm that can schedule any set of paths in $O(c+d)$ steps using constant size queues. We suspect that finding such an algorithm (if one exists) will be a challenging task. Our negative suspicions are derived from the fact that we can construct counterexamples to most of the simplest on-line algorithms. In other words, for several natural on-line algorithms (including the algorithm described in Section 3) we can find packet paths for which the algorithm will construct a schedule using substantially more than $\Omega(c+d+\log N)$ steps. Several of the counterexamples are included in Section 5.

2 An $O(c+d)$ off-line algorithm

In this section we show that for any set of paths with maximum congestion c and maximum distance d, in any network, there is a schedule of length $O(c+d)$ in which at most one packet traverses each edge of the network at each step, and at most $O(1)$ packets wait in each queue at each step. In our routing network model, all packets are stored in queues at the ends of edges. At each time step a packet either waits in a queue or traverses an edge and enters the queue at the end of that edge. We assume that at the beginning and end of the routing there is one packet in each queue. A *schedule* for a set of packets simply specifies at each time step which packets move and which wait.

Our strategy for constructing an efficient schedule is to make a succession of refinements to the "greedy" schedule, S_1, in which each packet moves at every step until it reaches its final destination. The length of S_1 is only d, but it does not meet the requirement that at most one packet traverses each edge of the network at each step, since as many as c packets may use an edge in a single step. Each refinement brings us closer to meeting this requirement by bounding the congestion within smaller and smaller frames of time. A *T-frame* is a sequence of T consecutive time steps. The *frame congestion*, C, in a T-frame is the largest number of packets that traverse any edge in the frame. The *relative congestion*, R, in a T-frame is the ratio C/T of the congestion in the frame to the size of the frame. A refinement transforms a schedule S_i with relative congestion at most $r^{(i)}$ in any frame of size $I^{(i)}$ or greater into a schedule S_{i+1} with relative congestion at most $r^{(i+1)}$ in any frame of size $I^{(i+1)}$ or greater, where $r^{(i+1)} \approx r^{(i)}$ and $I^{(i+1)} << I^{(i)}$. (For ease of notation, we use I and r in place of $I^{(i)}$ and $r^{(i)}$.) We shall assume without loss of generality that $c = d$. Thus, at the start, the relative congestion in a d-frame of S_1 is at most 1. After a series of $j = O(\log^* d)$ refinements, we obtain a schedule S_j with relative congestion $O(1)$ in every frame of size k_0 or greater, where k_0 is some constant. From S_j it is straightforward to construct a schedule of length $O(c+d)$ in which at most one packet traverses each edge of the network at each step, and at most $O(1)$ packets wait in each queue at each step.

In the ith refinement, schedule S_i is broken into *blocks* of $2I^3 + 2I^2 - I$ consecutive time steps. Each block is rescheduled independently. For each block, each packet is assigned a random delay chosen independently and uniformly from 1 to I. A packet assigned

a delay of x must wait for x steps at the beginning of the block. In order to bound the queue size and length of our final schedule, it is crucial that we maintain the invariant that in schedule S_{i+1} every packet waits at most once every $I^{(i)}$ steps. Thus, instead of delaying the packet for x consecutive steps at the beginning of the block, we insert one delay every I steps in the first xI steps of the block.[1] A packet that is delayed x steps reaches its destination at the end of the block by step $2I^3 + 2I^2 - I + x$. Since some packet may have delay $x = I$, the rescheduled block must have length $2I^3 + 2I^2$. In order to independently reschedule the next block, the packets must reside in exactly the same queues at the end of the rescheduled block that they did at the end of the block of S_i. Since some packets arrive early, they must be slowed down. Thus, a delay is inserted every I steps in the last $I(I - x)$ steps of the block. Note that at the beginning of the first block and end of the last block, it is not necessary to separate the delays by I steps.

After adding delays to S_i, the congestion may have increased in the I^2 steps at the beginning and end of each block. The following lemma shows that by increasing the frame size from I to I^2 we can bound the relative congestion in these regions.

Lemma 1 *The relative congestion in any frame of size I^2 or greater is at most $r(1 + 1/I)$.*

Proof: After the delays are inserted, a packet can use an edge in a T-frame if it used the edge in the frame or in any of the I steps before the frame in S_i. Thus, at most $r(T + I)$ packets can use an edge in the T-frame. For $T \geq I^2$, the relative congestion is at most $r(1 + 1/I)$. ∎

We now show that there is some way of choosing the delays so that in between the first and last I^2 steps we can decrease the frame size substantially without increasing the congestion much. The proof makes use of two lemmas. The first is used to bound the relative congestion over a wide range of frame sizes. The second is the quintessential tool of the probabilistic method: the Lovasz local lemma [22, pp. 57–58].

Lemma 2 *In any schedule, if the relative congestion in every frame of size T to $2T-1$ is at most R then the relative congestion in any frame of size T or greater is at most R.*

Proof: Consider a frame of size T', where $T' > 2T-1$. The first $(\lfloor T'/T \rfloor - 1)T$ steps of the frame can be broken into T-frames, each with relative congestion R. The remainder of the T'-frame consists of a single frame of size between T and $2T - 1$ steps in which the relative congestion is also at most R. ∎

Lemma 3 (Lovasz) *Let $A_1, \ldots, A_m$ be a set of "bad" events each occurring with probability p. Suppose that every bad event depends on at most b other bad events (i.e., every bad event is mutually independent of some set of $m - b$ other bad events). If $4pb < 1$, then the probability that no bad event occurs is nonzero.* ∎

With these lemmas in hand, we can proceed with the proof.

Lemma 4 *There is some way of choosing the packet delays so that in between the first and last I^2 steps of a block, the relative congestion in any frame of size $I_1 = \log^2 I$ or greater is at most $r_1 = r(1 + \epsilon_1)$, where $\epsilon_1 = O(1)/\sqrt{\log I}$.*

Proof: With each edge we associate a bad event. For edge e, a bad event occurs when more than $r_1 T$ packets use e in any T-frame for T in the range I_1 to $2I_1 - 1$. To show that no bad event occurs, we need to bound both the dependence of the bad events and the probability that an individual bad event occurs.

We first bound the dependence. At most $r(2I^3 + 2I^2 - I)$ packets use an edge in the block[2]. Each of these packets travels through at most $2I^3 + 2I^2 - I$ other edges in the block. As we shall see later, it will always be true that $r = r^{(i)} = O(1)$. Thus a bad event depends on $b = O(I^6)$ other bad events.

Now let us compute an upper bound on the probability, p_1, that more than $r_1 I_1$ packets use an edge in a particular I_1-frame. Since a packet may be delayed up to I steps before the frame, any packet that uses e in the frame or in any of the I steps before the frame in S_i may use e after the delays are inserted into S_i. Thus, there are at most $r(I + I_1)$ packets that can use e in the frame. For each of these the probability that the packet uses e in the frame after being delayed is at most (I_1/I). If we assume that no packet uses an edge more than once, then these probabilities are independent. Thus, the probability p_1 that more than $r_1 I_1$ packets use the frame is at most

$$p_1 \leq \sum_{k=r_1 I_1}^{r(I+I_1)} \binom{r(I + I_1)}{k} (I_1/I)^k (1 - I_1/I)^{r(I+I_1)-k}.$$

Let $r_1 = r(1+\epsilon_1)$. Using the inequalities $(1+x) \leq e^x$, $\ln(1 + x) \geq x - x^2/2$ for $0 \leq x \leq 1$, and $\binom{a}{b} \leq (ae/b)^b$ for $0 < b < a$, we have

$$p_1 \leq O(e^{-rI_1\epsilon_1^2(1/2-\epsilon_1/2-I_1/\epsilon_1^2 I-2I_1/\epsilon_1 I)}).$$

[1] Before the delays for schedule S_{i+1} have been inserted, a packet is delayed at most once in each block of S_i. Prior to inserting each new delay into a block, we check if it is within $I^{(i)}$ steps of the single old delay. If the new delay would be too close to the old delay, then it is simply not inserted. The loss of a single delay in a block has a negligible effect on the probability calculations in the lemmas that follow.

[2] Throughout the following lemmas we make references to quantities such as rI packets or $\log^4 I$ time steps, when in fact rI and $\log^4 I$ may not be integral. Rounding these quantities to integer values when necessary does not affect the correctness of the proof. For ease of exposition, we shall henceforth cease to consider the issue.

For $I_1 = \log^2 I$ and $\epsilon_1 = k_1/\sqrt{\log I}$, we can ensure that $p \leq 1/I^{k_2}$, for any constant $k_2 > 0$ by making constant k_1 large enough.

Next we need to bound the probability p_2 that more than $r_1 I_1$ packets use e in any I_1-frame of the block. There are at most $O(I^3)$ I_1-frames. Thus $p_2 \leq O(I^3)p_1$. By making the constant k_2 large enough, we can ensure that $p_2 \leq 1/I^{k_3}$, for any constant $k_3 > 0$.

The calculations for frames of size $I_1 + 1$ through $2I_1 - 1$ are similar. There are at most $O(I^3)$ frames of any one size, and $2I_1$ frame sizes between I_1 and $2I_1 - 1$. By adjusting the constants as before, we can guarantee that the probability p that more than $r_1 T$ packets use e in any T-frame for T between I_1 and $2I_1 - 1$ is at most $1/I^{k_4}$ for any constant $k_4 > 0$.

Finally, since a bad event depends on only $b = O(I^6)$ other bad events, we can make $4pb < 1$ by making k_4 large enough. By the Lovasz local lemma, there is some way of choosing the packet delays so that no bad event occurs. ∎

Although the frame size in the center of each block has decreased, it has increased from I to I^2 in the first and last I^2 steps of the block. To decrease the frame size in these regions, we move the block boundaries to the centers of the blocks. Now each block of size $2I^3 + 2I^2$ has a "fuzzy" region of size $2I^2$ in its center in which the relative congestion in any frame of size I^2 or greater is $r(1 + 1/I)$. In the I^3 steps before and after the fuzzy region, the relative congestion in any frame of size I_1 or greater is r_1. To reduce the frame size in the fuzzy region, we assign a random delay from 1 to I^2 to each packet. A packet with delay x waits once every I^3/x steps in the I^3 steps before the fuzzy region and once every $I^3/(I^2 - x)$ steps in the I^3 steps after the region. The rescheduled block now has size $2I^3 + 3I^2$.

We now show that there is some way of inserting delays into the schedule before the fuzzy region that both reduces the frame size in the fuzzy region, and does not increase either the frame size or the relative congestion before the fuzzy region by much. A similar analysis holds after the fuzzy region.

Lemma 5 *There is some way of choosing the packet delays so that between steps $I \log^3 I$ and steps I^3, the relative congestion in any frame of size I_1 or greater is at most $r_2 = r(1+\epsilon_2)$, where $\epsilon_2 = O(1)/\sqrt{\log I}$, and so that in the fuzzy region the relative congestion in any frame of size I_1 or greater is at most $r_3 = r(1 + \epsilon_3)$, where $\epsilon_3 = O(1)/\sqrt{\log I}$.*

Proof: Since no delays are inserted into the fuzzy region, the proof that the frame size has been reduced in the fuzzy region is analogous to the proof of the previous lemma.

Before the fuzzy region, the situation is more complex. By the kth step, $0 \leq k \leq I^3$, a packet with delay x has waited xk/I^3 times. Thus, the delay of a packet at the kth step varies essentially uniformly from 0 to $u = k/I$. For $u \geq \log^3 I$, or equivalently, $k \geq I \log^3 I$, we can show that the relative congestion in any frame of size I_1 or greater has not increased much.

The proof uses the Lovasz local lemma as before. The calculation for the dependence is unchanged. The probability p_2 that more than $r_2 I_1$ packets use an edge e in a particular I_1-frame is given by

$$p_2 \leq \sum_{s=r_2 I_1}^{r_1(I_1+u)} \binom{r_1(I_1+u)}{s} (I_1/u)^s (1 - I_1/u)^{r_1(I_1+u)-s}.$$

Using the same inequalities as before, we have

$$p_2 \leq O(e^{-r_1 I_1 \epsilon_2^2 (1/2 - \epsilon_2/2 - I_1/\epsilon_2^2 u - 2I_1/\epsilon_2 u)}).$$

For $I_1 = \log^2 I$, $u \geq \log^3 I$, it suffices that $\epsilon_2 = O(1)/\sqrt{\log I}$. ∎

For steps 0 to $I \log^3 I$, we use the following lemma to bound the frame size and relative congestion.

Lemma 6 *The relative congestion in any frame of size I_2 or greater between steps 0 and $I \log^3 I$ is at most r_4, where $I_2 = \log^4 I$ and $r_4 = r_1(1 + 1/\log I)$.*

Proof: The proof is similar to that of Lemma 1. ∎

We have now completed our transformation of schedule S_i into schedule S_{i+1}. Let us review the relative congestion and frame sizes in the different parts of a block of S_{i+1}. Between steps 0 and $I \log^3 I$, the relative congestion in any frame of size I_2 or greater is at most r_4. Between this region and the fuzzy region, the relative congestion in any frame of size I_1 or greater is at most r_2. In the fuzzy region, the relative congestion in any frame of size I_1 or greater is at most r_3. After the fuzzy region, the relative congestion in any frame of size I_1 or greater is again r_2, until step $2I^3 + 3I^2 - I \log^3 I$, where the relative congestion in any frame of size I_2 or greater is r_4. For the entire block it is safe to say that the relative congestion in any frame of size $I^{(i+1)} = \log^4 I$ or greater is at most $r^{(i+1)} = r(1 + O(1)/\sqrt{\log I})$.

The following theorem shows that by repeatedly applying this refinement step, we an construct an asymptotically optimal schedule.

Theorem 7 *For any set of paths with maximum congestion c and maximum distance d, there is a schedule of length $O(c+d)$ in which at most one packet traverses each edge of the network at each step, and at most $O(1)$ packets wait in each queue at each step.*

Proof: Without loss of generality, assume $c = d$.

We begin by assigning each packet a random delay chosen uniformly from 0 to d at the beginning of the greedy schedule S_1. Using the Lovasz local lemma, we can show that there is some way of choosing the

delays so that in the resulting schedule S_2, the relative congestion is at most $r^{(1)} = O(1)$ in any frame of size $I^{(1)} = \log d$ or greater.

Next, we repeatedly use the refining algorithm to reduce the frame size. The relative congestion $r^{(i+1)}$ and frame size $I^{(i+1)}$ for schedule S_{i+1} are given by the recurrences

$$r^{(i+1)} = \begin{cases} O(1) & i = 1 \\ r^{(i)}(1 + O(1)/\sqrt{\log I^{(i)}}) & i > 1 \end{cases}$$

and

$$I^{(i+1)} = \begin{cases} \log d & i = 1 \\ \log^4 I^{(i)} & i > 1 \end{cases}$$

which have solutions $I^{(j)} = O(1)$ and $r^{(j)} = O(1)$, where $j = O(\log^* d)$.

We have not explicitly defined the values of r and I for which the recursion terminates. However, in many places we implicitly use the fact that I is sufficiently large or r is sufficiently small that certain inequalities hold. The recursion terminates when the first of these inequalities fails to hold. When this happens, one of r or I is $O(1)$, which implies that the other is also.

Since a packet waits at most once every $I^{(i)}$ steps in S_i, it waits at most once every $\Omega(1)$ steps in S_j, which implies both that the queues in S_j cannot grow larger than $O(1)$ and that the total length of S_j is $O(d)$.

Schedule S_j almost satisfies the requirement that at most one packet traverse each edge in each step. By simulating each step of S_j in $O(1)$ steps we can meet this requirement with only a factor of 2 increase in the queue size and a factor of $O(1)$ increase in the running time. ∎

Why is this proof so complicated? Using the same basic ideas, it is possible to construct in a much simpler fashion a schedule of length $2^{O(\log^* d)}d$ that uses queues of size $O(\log d)$. Unfortunately, removing the $2^{O(\log^* d)}$ factor seems to require delving into second order terms in the probability calculations, and reducing the queue size to $O(1)$ mandates great care in spreading delays out over the schedule.

3 On-line algorithms

3.1 An $O(c + d \log n)$ on-line algorithm

By applying the type of probabilistic analysis used in Section 2, it is fairly straightforward to schedule any set of n paths in $O(c + d \log n)$ steps with queues of size $O(\log n)$. We simply delay the start of each packet by a random amount that is chosen uniformly from $[1, \frac{c}{\log n}]$, and then route all the packets forward in a synchronized fashion. More precisely, we introduce the initial delays and then consider the unconstrained schedule without regard for the rule that at most one packet traverse any edge in a single step. With high probability, no more than $O(\log n)$ packets will want to traverse any edge at any step of the unconstrained schedule. Hence we can simulate each step of the unconstrained schedule with $O(\log n)$ steps of a legitimate schedule. The final schedule consumes $O((d + \frac{c}{\log n}) \log n) = O(c + d \log n)$ steps to complete the routing and uses $O(\log n)$ size queues.

3.2 An $O(c+d+\log n)$ on-line algorithm for leveled networks

Consider any set of n leveled paths spanning d levels with congestion c. For simplicity, we will think of the packets as being distinct (i.e., no combining will be allowed) although our analysis can easily be extended to the case where arbitrary combining takes place. We will allow up to c packets to originate at the same node and to end at the same node. For this purpose, we will allow queues of size c at the first and last levels, but will restrict queues in the interior levels to have constant size q. The value of q can be any integer (including 1), and will affect the overall routing time by a constant factor. We will also assume that the underlying network has indegree and outdegree 2, although the result can easily be extended to networks with any constant degree. In what follows, we show how to route all the packets in $O(d + c + \log n)$ steps with high probability without overflowing any queue.

The algorithms for scheduling the packets is identical to Ranade's algorithm except that we select random keys by which the packets are ordered instead of ordering based on destination address, as in [21]. In particular, each packet is assigned a random key and a packet is routed through a node only after all the other packets with lower keys that are destined to be routed through the same node have done so. Queues are placed at the end of each edge and a packet advances forward only if there is already room for the packet in its next queue. For simplicity we will assume that the queue size is at least two, so that once a queue contains a packet, it does not become empty until it transmits an end-of-stream signal. With minor modifications, the analysis can be made to work with queues of size one. To keep things moving, ghost messages are sent along each edge that is not transmitting a packet. The ghost message provides the best lower bound known by the node for the size of the key of the next packet to be sent. Ghost messages allow a processor to send a packet forward from one incoming edge without having to wait for actual packets (if any) on the other incoming edges (provided, of course that the ghost messages on these incoming edges indicate that any such packets would have to have higher keys). Ghost messages are saved only if they arrive at the head of the queue.

To prove that the algorithm completes the routing in $O(c + d + \log n)$ steps, we use the same delay path argument as Ranade [18] (which, in turn is quite similar to the ones used by Aleliunas [2] and Upfal [24]), but we simplify the counting part of the analysis. The simplified counting has the additional nice feature that

it allows the interior queue sizes to be as small as one, which was not possible with Ranade's original analysis. We provide a sketch of the argument in what follows. The complete proof will be included in the full draft of the paper.

If some packet is delayed by w steps during the course of the routing, then there is a delay path of length l that coincides with w packet paths arranged in order of decreasing key size (working backward). If f is the number of forward edges in the delay path, then $w \geq qf/2$ and $l = d + 2f \leq d + \frac{4w}{q}$. The number of possible delay paths is at most

$$n4^l \binom{l+w}{w} (2c)^w$$

since there are n places that the path can start, 4^l ways that it can continue, $\binom{l+w}{w}$ ways of locating the points of incidence with w packet paths, and $(2c)^w$ ways to pick packets at the points of incidence.

If the random keys are chosen from $[1, w]$, then the probability that the w keys for any delay path are in the right order is at most $\binom{2w}{w}/w^w$. Hence, the probability that there is a delay path corresponding to w delay is at most

$$\frac{n4^l \binom{l+w}{w} (2c)^w \binom{2w}{w}}{w^w} \leq \left(\frac{2^{\frac{\log n + 2d}{w} + \frac{12}{q} + 3} ce^2}{w}\right)^w.$$

For $w = \Omega(d + c + \log n)$, this probability can be made arbitrarily small, even if $q = 1$.

Note that in the case that $w = \Theta(\log n)$, the preceding argument requires only $O(\log n \log\log n)$ bits of randomness, since the keys lie in the range $[1, w]$, and we only require that sets of w keys be independent.[3]

4 Applications

It is straightforward to apply the algorithm described in Section 3 to route packets on arrays and butterflies. For two-dimensional arrays, the paths of the packets are selected greedily with each packet first traveling to the correct row, and then to the correct column. For arrays of higher dimension and butterflies, random path selection works fine, and the resulting time bounds are within a constant factor of optimal. Some care must be taken to get around the queues of size c at the first and last levels of the network, but this is not difficult to do.

It is not so clear how to apply the algorithm from Section 3 to route on networks such as the shuffle-exchange or the deBruijn graphs, however. The reason is that they do not have an apparent leveled structure. Nevertheless, we can still obtain a good routing algorithm for these networks by identifying a leveled-like structure in a large portion of the shuffle-exchange graph. The details are included in Section 4.1.

In Section 4.2, we show how to adapt the on-line algorithm to efficiently route on fat-trees[13], thus providing the first examples of area and volume-universal networks with slowdown $O(\log N)$.

4.1 Routing on the shuffle–exchange graph

In this section, we will show how to apply the techniques of Section 3 to obtain a randomized routing algorithm on the shuffle-exchange graph. It works for any N packet routing problem with at most one packet starting at any node and runs in $O(\log N)$ steps using constant size queues.

The N-node *shuffle-exchange graph* is defined for every N which is a power of two. Each node of the $(N = 2^k)$-node shuffle exchange graph is associated with a unique k-bit binary string $a_{k-1}...a_0$. Two nodes w and w' are linked via a *shuffle* edge if w' is a left or right cyclic shift of w. Two nodes w and w' are linked via an *exchange edge* if w and w' differ in the least significant bit, a_0.

A node of the shuffle-exchange graph is *good* if it has a unique longest cyclic substring of zeros of length greater than $\log\log N - 1$, and it is not node 0. ($w = 0...0$ is not a good node.) Any node that is not a good node is *bad*.

We group the nodes into *necklaces* consisting of nodes that are cyclic shifts of one another. A necklace consists entirely of good nodes or entirely of bad nodes since the cyclic length of any substring of zeros is unchanged by a cyclic shift. Each good node necklace consists of $\log N$ nodes since each cyclic shift is different due to the unique longest string of zeros.

We route mainly by using the good node necklaces as a leveled structure, thus we associate bad nodes with good necklaces. We show that at most $3\log N$ bad nodes are associated with any good necklace.

Consider that there are three types of bad nodes,

1. nodes having a longest string of zeros of length less than $\log\log N - 1$,
2. nodes with more than one group of longest zeros,
3. and node $w = 0\ldots0$.

A bad node of type 1 is mapped to a good node by making the least significant bit a one and the $\log\log N - 1$ most significant bits to zeros. This associates the bad node with the lexicographically minimum node of a good necklace, since after the transformation, the highest order bits are composed of the longest string of zeros. Only bad nodes which differ from a good necklac 's lexicographically minimum node in at most $\log\log N$ bits are mapped to it, thus at

[3] The use of the range $[1, w]$ for the random keys was suggested to us by Ranade [19]. Essentially the same range is used in [21]. The constants in the running time can be reduced by increasing the range to $[1, n^3]$. In this case, the number of random bits required is $O(\log^2 n)$.

most $2^{\log\log N} = \log N$ type 1 bad nodes are associated with any good necklace.

We map a bad node of type 2 to a good necklace by mapping a bad necklace to a good node necklace. We take the lexicographically minimum node in a type 2 bad node necklace and extends its leading group of zeros by exactly one zero. All the bad nodes in this necklace are associated with the specified good necklace.

To assess the number of bad nodes associated with a good necklace by this operation, we consider the lexicographically minimum node in the good necklace and notice that only bad necklaces whose minimum node differs in the last bit of the leading block of zeros and possibly differs in the bit after that is mapped to that necklace. Thus, at most two bad necklaces are associated with any good node and thus only $2\log N$ bad nodes of type 2 are associated with any good necklace.

Finally, node 0 is simulated by node 1. Note that no bad nodes of type 1 or 2 are associated with node 1's necklace.

So in all, at most $3\log n$ bad nodes are associated with any good necklace. Recalling that all good necklaces contain $\log N$ nodes, we have $N/4$ of the shuffle-exchange nodes being good. We use these nodes to perform the bulk of the routing. The basic idea is that we deterministically route packets from bad nodes to good, then use a randomized routing algorithm to route between good nodes, and finally deterministically route packets from good nodes to bad. We proceed by defining a leveled network on the good nodes that the shuffle exchange graph can easily simulate with constant overhead. For any routing problem on the good nodes we construct paths in the leveled network with congestion and distance $O(\log N)$ with high probability. By applying the analysis of Section 3, we can then complete the routing in $O(\log N)$ steps with high probability using constant sized queues. We conclude by detailing the routing between good and bad nodes.

4.1.1 A leveled network

For each necklace of good nodes, we pick the lexicographically minimum node to be the representative node for the necklace. We denote each good node by its necklace's representation plus a line under the least significant bit, which we refer to as the *current* bit . For example, node 100011 would be written as $000\underline{1}11$. We define the *level* of the good node to be the position of the underline counting from the left. For example, $000\underline{1}11$ is in level 4. (Note that the representative node is in level $\log N - 1$.)

The leveling of the nodes just described induces a leveling of the shift edges but does not necessarily induce a leveling of the exchange edges. An exchange edge even between good nodes may create a new longest group of zeros by joining two groups of zeros and thus connect two levels which are very far apart. To overcome this difficulty we assume the graph contains *flip* edges. A flip edge links nodes w and w' if both w and w' are good nodes with $w = a_{k-1}...\underline{a_j}...a_0$ and $w' = a_{k-1}...\overline{a_j}\underline{1}..a_0$ and a_j is not in the leading block of zeros of w's representative node.

Note that flip edges extend a group of zeros by at most one. Thus no flip edge can create a new leading group of zeros, since it can only grow the shorter non-leading group of zeros to be as big as the leading group. But then it would lead to a bad node, i.e., a node having two different longest groups of zeros. This is a contradiction since flip edges only occur between good nodes by definition. Thus flip edges are leveled.

It is easily shown that the operation of the flip edges can be simulated by the shuffle-exchange graph with only a constant slowdown; each flip edge is composed of an exchange edge, a shuffle edge, and possibly another exchange edge.

We denote by network A the network composed of the good nodes, the shuffle edges, excluding the shuffle edges from level $\log N - 1$ to 0, and the flip edges.

Note that in network A, from any level 0 good node we can reach any necklace with a longest string of zeros having the same or greater length by correcting bits starting from the end of the leading block of zeros. In fact, we wish to be able to get from the level 0 node of a good node necklace to *any* other good node necklace.

Thus we append a mirror image of the good nodes with flip and shuffle edges to itself so that we can reach necklaces with fewer zeros. The leveling is extended in the natural manner. We call this whole thing network AA^r, and note that network A can easily simulate it.

We denote by network L, the network consisting of the shuffle edges on the good nodes again excluding shuffle edges from level $\log N - 1$ to level 0. Our method of path selection consists of routing from a good node to its level zero node, then routing to a random intermediate necklace, then routing to the destination necklace, and finally routing to the appropriate good node. Thus, the leveled network we route paths in is composed of network L, network AA^r, another network AA^r, followed by netwrok L. We extend the leveling in the natural manner and note that network A can easily simulate the whole thing.

4.1.2 Path selection and congestion

We assume that at the start of the routing there at most b packets starting at any good node. The value of b depends on the number of bad nodes mapped to any good node which we showed is small.

For each packet we choose its path by uniformly choosing a random necklace to route through before going to its final destination. So the path for a packet consists of a path through L to node 0 of its necklace, the path through AA^r to its random intermediate necklace, the path through the second AA^r to its destination necklace, and a path through the second L to the proper node of the necklace.

We show that this method yields paths with congestion $O(\log N)$ with high probability. That is, we show that the probability of any edge being used by more than $c \log N$ packets is $O(\frac{1}{N^{c-1}})$ for some constant c.

We observe that for the paths in the copies of L, we have congestion $b \log N$, since at most $b \log N$ packets start or end in any good necklace. By symmetry we claim that the analysis of the path portions in both copies of AA^r is the same. Finally we recall that in AA^r, we route packets going to necklaces with same or more zeros to the appropriate necklace in network A and straight across network A^r, we route the other packets straight across in network A and use A^r to route to the proper necklace. We will show that any destination necklace gets $O(c \log n)$ packets with high probability, so the straight across portion of the paths should not be a problem. To finish, we give the analysis of the congestion due to packets in just network A, and claim that the arguments will hold be symmetry for A^r.

Consider an edge in the first copy of network A. In this half, packets going to necklaces with fewer zeros are routed straight across their starting necklace. There are at most $b \log N$ of these, so without loss of generality we ignore them. Suppose that e traverses levels m and $m+1$. Let x be the number of zeros in the necklace to which e goes. If $m < x$, then no packet from any other necklace uses e, since we only map to a necklace via flip edges after its longest string of zeros. Otherwise, we consider the number of packets from other necklaces that can use e. We know that only packets from at most 2^l other necklaces with $l = m - \log\log N$ could have used e since at most l bits could have changed by level $m+1$. Thus the number of packets that can use e is at most $2^l b \log N$ since each necklace starts with at most $b \log n$ packets. The probability that a specific packet uses e, is the number of necklaces that can be reached using e, at most $2^{\log N - \log\log N - l}$ (i.e., necklaces which match e's necklace in the first $l + \log\log n$ bits), divided by the total number of good necklaces, at least $b' \frac{n}{\log N}$ (since a constant fraction of the nodes are good and there are $\log N$ nodes in a necklace), which is just $\frac{2^{-l}}{b'}$. Thus the probability that more than $c \log n$ packets use e can be written

$$\sum_{s=c\log N}^{s=N} \binom{2^l b \log N}{s} \left(\frac{2^{-l}}{b'}\right)^s$$

The first factor in each term gives the number of ways to choose the packets. The second is the probability that all these packet use e. The sum is bounded by $O(\frac{1}{N^c})$ if we choose $c > \frac{2b'}{b}$. Thus, the probability that any of the $O(N)$ edges of this stage has congestion more than $c \log N$ is then clearly $O(\frac{1}{N^{c-1}})$. For large enough c, this gives the desired high probability result $O(\frac{1}{N})$. This argument, also provides the proof that any random destination necklace receives $c \log n$ packets with high probability, since we need only consider the congestion on the edge from A to A^r.

We are finished showing how to route packets between the good nodes in a leveled fashion with path congestion and distance $O(\log N)$ with high probability. Thus, by the arguments of section 3 we can solve any routing problem on the good nodes in time $O(\log N)$ using constant sized queues.

4.1.3 Packets from bad nodes

In this section we show how to deterministically route a bad node's packet to its associated good node.

Recall that we associated a bad node of type 1 with the necklace represented by a one in the least significant or current bit plus $\log\log N - 1$ zeros in the most significant bits. We route these packets in the shuffle exchange graph by flipping the current bit to a one and flipping $\log\log N - 1$ bits to the right to zeros. Thus we map a bad node to a good necklace at its level $\log\log N$ node.

For any necklace, we have a binary tree, the leaves of which are mapped to the necklace. Each level of the tree corresponds to one of the $\log\log N$ bits that were flipped. Therefore, we can route packets from the binary tree leaves to the necklace, and distribute them along the necklace deterministically. This is easily done in $\log N$ time with constant queues. The routing from the necklace to the tree is equally trivial. But, we need to ensure that traffic from the separate binary trees does not interfere too much. This is easy since any bad node is in at most two binary trees; in at most one as a leaf since any node is mapped to exactly one good node, and in at most one as an internal node since the number of zeros between the current node and the closest one to the left determines a unique level and the rest of the bits determine a unique tree.

To finish, we consider unleveled nodes of type 2. These are nodes without a unique longest string of zeros. Here we extend one of the groups of zeros by one zero, making sure not to join two groups of zeros by inserting a one if necessary, i.e., mimicking the flip operation. For any good necklace whose representative is $0^k1\ldots$ only the necklaces represented by $0^{k-1}10\ldots$ and $0^{k-1}11\ldots$ can be mapped to it. Again, at most two bad necklaces are associated with any good necklace.

For each packet in such a bad necklace we route it through the node connecting it to the appropriate good necklace. We perform this movement by pipelining the packets through the edge which connects the two necklaces. We see that this mapping maps at most one packet from the bad necklace to a node in the good necklace. Since we are basically routing on linear arrays of length at most $2\log N$, $2\log N$ steps suffice to route the packets appropriately. $4\log N$ steps is sufficient to route the packets from two bad necklaces.

This finishes the description of the maps to and from all the unleveled nodes except for the node $w = 0$, which is easily routed to node 1.

4.2 Construction of area and volume-universal networks

In this section we construct a class of point-to-point networks that are *area-universal* in the sense that a network in the class with N processors has area $O(N)$ and can, with high probability, simulate in $O(\log N)$ steps each message-step of any shared-bus network of area $O(N)$. The simulation is optimal because a point-to-point network may require $\Omega(\log N)$ steps to simulate one step of a shared-bus network. The networks are based on the fat-trees of Greenberg and Leiserson [5] and the simulation uses the message routing algorithm from Section 3.

In a fixed-connection network, processors communicate via wires. Each processor has a bounded number of read and write pins. In a point-to-point network, each wire connects one read pin with one write pin. In each message-step, the processor with the write pin may transmit a message of $O(\log N)$ bits to the processor with the read pin. In a shared-bus network, a wire may connect many read and write pins. Such a wire is called a bus. In each message-step, any processors wishing to send messages make them available on their write pins. Then the messages at the write pins of each wire are *combined* by some simple rule to form a single message. Combining is assumed to require a single message-step, regardless of the number of messages combined or the rule used.

Leiserson was the first to display a class of fixed-connection networks that could efficiently simulate any other network of the same area or volume. In [13] he showed that a fat-tree of area $O(N)$ can simulate in $O(\log^3 N)$ bit-steps each bit-step of any fixed connection network of area $O(N)$. The simulation used an off-line routing algorithm for fat-trees. On-line routing algorithms were later developed by Greenberg and Leiserson [5] and Park [16]. None of these routing algorithms are capable of combining messages to the same destination. As a consequence, no scheme for simulating shared-bus networks was known until now. A network that can simulate in $O(1)$ steps each step of any shared-bus network area of equal area was presented in [15]. However, the connections in this network are not fixed, but instead processors communicate via reconfigurable busses.

A fat-tree network is shown in Figure 1. Its underlying structure is a complete 4-ary tree. Each edge in the 4-ary tree corresponds to a pair of oppositely directed groups of wires called *channels*. The channel directed from the leaves to the root is called an up channel; the other is called a down channel. The *capacity* of a channel c, $\mathrm{cap}(c)$, is the number of wires in the channel. We call the tree "fat" because the capacities of the channels grow by a factor of 2 at every level. A fat-tree of height m has $M^2 = 2^{2m}$ leaves and $M = 2^m$ vertices at the root.

It will prove useful to label the switches at the top and bottom of each channel. Let the level of a switch be its distance from the leaves. Suppose a channel c connects $cap(c)/2 = 2^l$ switches at level l with $cap(c) = 2^{l+1}$ switches at level $l+1$. Give the switches at level l labels 0 through $2^l - 1$ and the switches at level $l+1$ labels 0 through $2^{l+1} - 1$. Then switch k at level l is connected to switches k and $k + 2^l$ at level $l+1$. The following lemma relates the labels of the switches on a message's path from a leaf to the root.

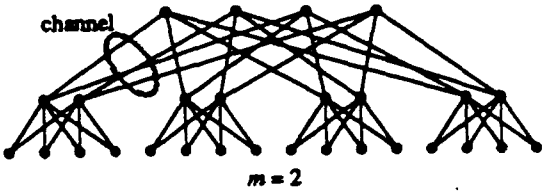

Figure 1: A fat-tree.

Lemma 8 *There is a unique shortest path from any leaf to a switch labeled k at the root, for $0 \le k \le M-1$, and that path passes through a switch labeled $k \bmod 2^l$ at level l, for $0 \le l \le m$.* ∎

For a set Q of messages to be delivered between the leaves of the fat-tree, we define the *load* of Q on a channel c, $\mathrm{load}(Q, c)$, to be the number of destinations of messages in Q for which at least one message must pass through c. Note that even if many messages with the same destination must pass through a channel, that destination contributes at most one to the load of the channel. We define the *load factor* of Q on c, $\lambda(Q, c)$, to be the ratio of the load of Q on c to the capacity of c, $\lambda(Q, c) = \mathrm{load}(Q, c)/\mathrm{cap}(c)$. The load factor on the entire network, $\lambda(Q)$ is simply the maximum load factor on any channel $\lambda(Q) = \max_c \lambda(Q, c)$. The load factor is a lower bound on the the number of steps required to deliver Q. We shall assume that $\lambda \le M^k$, where k is some fixed constant. We shall sometimes write λ to denote $\lambda(Q)$ when the set of messages to be delivered is clear from the context.

In a *leveled fat-tree* a switch at the top of an up channel at level l is connected to itself at the top of the corresponding down channel by a linear chain of switches of length $2(m-l)$. A message may only make a transition from an up channel to a down channel by traversing a chain. Thus all shortest paths between leaves in a leveled fat-tree have length $2m$. Note that the load of a set of messages on a channel of the leveled fat-tree is identical to the load on the corresponding channel in the fat-tree.

The path that a message for destination x in column $2m$ takes through a leveled fat-tree is determined by

the m-universal hash function[4]

$$\text{path}(x) = \left(\left(\sum_{i=0}^{m-1} a_i x^i \right) \bmod P \right) \bmod M,$$

where P is a prime number larger than the number of possible different destinations, and the $a_i \in Z_P$ are chosen at random off-line. A message with destination x follows up channels until it can reach x without using any more up channels. It then crosses over to a down channel via a chain, and follows down channels to x. Note that a message only passes through a channel if it must. Also, all messages with destination x that pass through channel c pass through switch $(\text{path}(x) \bmod \text{cap}(c))$ at the top of c and through switch $(\text{path}(x) \bmod (\text{cap}(c)/2))$ at the bottom of c.

The following lemma shows that we can use the scheduling algorithm from Section 3 to route messages in a fat-tree.

Lemma 9 *For any constant c_1, there is a constant c_2 such that the probability that the number of steps required to deliver a set Q of messages with load factor λ is more than $c_2(\lambda + \log M)$ is at most $1/M^{c_1}$.*

Proof: The paths of the messages are first randomized using the universal hash function path. With high probability, the resulting congestion is $c = O(\lambda + \log M)$. Each message travels a distance of $d = 2m = 2\log M$. The messages are then scheduled using the algorithm from Section 3. ∎

Let us now consider the VLSI area requirements [23] of fat-trees. A fat-tree with root capacity M and $\Theta(M^2)$ processors has a layout with area $O(M^2 \log^2 M)$ that is obtained by embedding the fat-tree in the tree of meshes[10]. The nodes of the tree of meshes in this layout are separated by a distance of $\lg M$ in both the horizontal and vertical directions. Thus, the $\Theta(\log M)$ space for the chain associated with each processor in the leveled fat-tree can be allocated without increasing the asymptotic area of the layout. (In fact, it is possible to attach a chain of size $O(\log^2 M)$ to each fat-tree node without increasing the area by more than a constant factor.) The leaves of the fat-tree are separated in the layout from each other by a distance of $\lg M$ in each direction. We can improve the density of processors without increasing the asymptotic area of the layout by connecting a $\lg M \times \lg M$ mesh of processors to each leaf. The resulting network has $\Theta(M^2 \log^2 M)$ processors and area $\Theta(M^2 \log^2 M)$. The N-processor network in this class has root capacity $\Theta(\sqrt{N}/\log N)$, $\Theta(N/\log^2 N)$ leaves, and area $\Theta(N)$.

The following theorem shows that this class of networks is area-universal.

Theorem 10 *With high probability, an N-processor point-to-point fixed-connection network U of area $\Theta(N)$ can simulate in $O(\log N)$ steps each step of any shared-bus fixed-connection network B of area $O(N)$.*

Proof: The processors of the shared-bus network B are mapped to the processors of the area-universal network U off-line using a recursive decomposition technique as in [13]. In each step, a wire of B is simulated by routing messages between the processors that it connects. At each level of the recursion at most $O(\text{cap}(c) \cdot \log N)$ wires connect the processors mapped below a channel c with the rest of the network. This property of the mapping ensures that the load factor of each set of messages used in the simulation of B is at most $O(\log N)$. At the bottom of the decomposition tree, a $O(\log N) \times O(\log N)$ region of the layout of B is mapped to each leaf of the fat-tree. The $O(\log N) \times O(\log N)$ mesh connected to the leaf in U simulates this region of B using standard mesh routing algorithms. ∎

The study of fat-tree routing algorithms that perform combining was motivated in part by an abstraction of the volume and area-universal networks called the distributed random-access machine (DRAM). A host of conservative algorithms for tree and graph problems for the exclusive-read exclusive-write (EREW) DRAM are presented in [14]. Recently we discovered conservative concurrent-read concurrent-write (CRCW) algorithms that require fewer steps for some of these problems. Until now, however, no efficient fat-tree routing algorithms that perform combining were known. The $O(\lambda + \log N)$ step routing algorithm presented here fills the void.

Only slight modifications to the area-universal fat-tree are necessary to make it volume universal[5]. The underlying structure of the volume-universal fat-tree is a complete 8-ary tree. Instead of doubling at each level, the channel capacities increase by a factor of 4. The tree has m levels, root capacity $M = 2^{2m}$, and $M^{3/2} = 2^{3m}$ leaves. The switches at the top of a channel at level l are labeled 0 through $4^l - 1$. Switch k at level l is connected to switches k, $k + 4^l$, $k + 2 \cdot 4^l$, and $k + 3 \cdot 4^l$ at level $l + 1$. A layout with volume $O(M^{3/2} \log^{3/2} M)$ for the fat-tree can be obtained by embedding it in the three-dimensional tree of meshes. As before, a chain of size $O(\log^{3/2} M)$ can be attached to each node of the fat-tree without increasing the asymptotic layout area and the density of processors can be improved by connecting a $\lg^{1/2} M \times \lg^{1/2} M \times \lg^{1/2} M$ mesh to each leaf.

5 Counterexamples to on-line algorithms

In this section we give examples where several on-line scheduling strategies do poorly. Based on these examples, we suspect that finding an on-line algorithm that can schedule any set of paths in $O(c + d)$ steps using constant size queues will be a challenging task.

In the first example, we describe an N-node network in which a set of packets with maximum congestion and maximum distance $O(1)$ requires $\Omega(\log^2 N / \log \log N)$

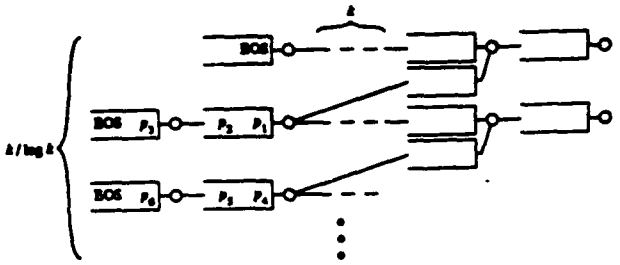

Figure 2: Example 1.

steps to be delivered using the strategy of Section 3. This example does not contradict the results of Section 3, since the network has $\Theta(\log^2 N)$ levels. However, it shows that reducing the maximum congestion and maximum distance below the number of levels will not necessarily improve the running time.

Observation 11 *For the strategy of Section 3, there is an N-node directed acyclic network of degree 3 and a set of paths with maximum congestion $c = 3$ and maximum distance $d = 3$ where the expected length of the schedule is $\Omega(\log^2 N / \log\log N)$.*

Proof: The network consists of many disjoint copies of the subnetwork pictured in Figure 2. The subnetwork is composed of $k/\log k$ linear chains of length k, where k shall later be shown to be $\Theta(\log N)$. The second node of each linear chain is connected to the second to last node of the previous chain by a diagonal edge. We assume that at the end of each edge there is a queue that can store 2 packets. Initially, the queue into the first node of each chain contains an (EOS) end-of-stream signal and one packet, and the queue into the second node contains two packets. A packet's destination is the last node in the previous chain. Each packet takes the diagonal edge to the previous chain and then the last edge in the chain. Thus, the length of the longest path is $d = 3$.

When the ranks $r_1, \ldots, r_{3k/\log k}$ of the packets $p_1, \ldots, p_{3k/\log k}$ are chosen so that $r_i < r_{i+1}$ for $1 \le i < 3k/\log k$, packet $p_{3k/\log k}$ requires $\Omega(k^2/\log k)$ steps to reach its destination. The scenario unfolds as follows. Packets p_1 and p_2 take a diagonal edge in the first two steps. These packets cannot advance until the EOS reaches the end of the first chain, in step k. In the meantime, ghosts with ranks r_1, r_2, and r_3, travel down the second chain, but packet p_3 blocks an EOS signal from traveling down the chain. Packets p_4 and p_5 are waiting for this EOS signal. They cannot advance until step $2k$. In this fashion, the delay is propagated down to packet $p_{3k/\log k}$.

A simple calculation reveals that the probability that $r_i < r_{i+1}$ for $1 \le i \le 3k/\log k$ is $1/2^{\Theta(k)}$. Thus, if we have $2^{\Theta(k)}$ copies of the subnetwork, we expect the ranks of the packets to be sorted in one of them. For the total number of nodes in the network to be N, we need $k = \Theta(\log N)$. In this case, we expect some packet to be delayed $\Omega(\log^2 N/\log\log N)$ steps in one copy of the subnetwork. ∎

It is somewhat unfair to say that the optimal schedule for this example has length $O(c + d) = O(1)$, since ghosts and EOS signals must travel a distance of $\Theta(\log N)$. However, even if the EOS signals are replaced by packets with the appropriate ranks, the maximum distance is only $O(\log N)$, and thus the optimum schedule has length $O(\log N)$.

The second example is quite general. The following observation shows that for any deterministic strategy in which the order in which packets are chosen to pass through a switch is independent of the future paths of the packets, there is a network and a set of paths with maximum congestion c and maximum distance d in which the schedule produced has length cd. This observation covers strategies such as giving priority to the packet that has spent the most (or least) time waiting in queues, and giving priority to the packet that arrives first at a switch. The network has the disadvantage of having degree c and size c^d.

Observation 12 *For any deterministic strategy in which the order in which packets are chosen to pass through a switch does not depend on the paths that the packets take after they pass through the switch, there is a network and a set of paths with congestion c and maximum distance d for which the schedule produced has length cd.*

Proof: We construct the example for congestion c and maximum distance d, $E(c,d)$, recursively. The network consists of c copies of the network for $E(c, d-1)$ feeding into a single edge e. For each copy of $E(c, d-1)$, the strategy schedules some packet to arrive at e last. We extend the path of this packet so that it traverses e in $E(c,d)$. The maximum distance of the new set of paths is d and the congestion c. The length of the schedule, $T(c,d)$, is given by the recurrence

$$T(c,d) = \begin{cases} T(c,d-1)+c & d > 1 \\ c & d = 1 \end{cases}$$

and has solution $T(c,d) = cd$. ∎

The third and fourth examples show that simple scheduling strategies fail even in much smaller networks.

Observation 13 *For the strategy in which the packet with the farthest distance left to travel or the farthest total distance to travel is given priority, there is an N-node network with diameter $O(\sqrt{N})$ and a set of paths with congestion $O(\sqrt{N})$ and maximum distance $O(\sqrt{N})$ for which the schedule produced has length $\Omega(N)$.*

Proof: The network consists of $\sqrt{N}/2$ linear chains. Chain i is composed of $\sqrt{N} + i$ nodes. It meets chain

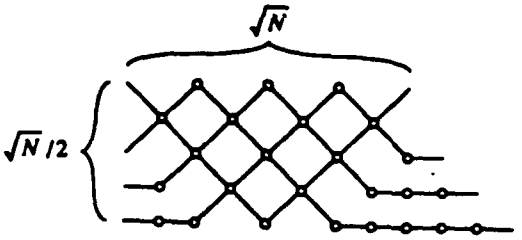

Figure 3: Example 3.

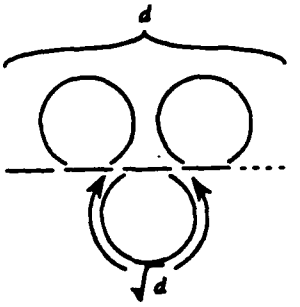

Figure 4: Example 4.

$i-1$ at node i and at every second node after that up to node $\sqrt{N}-i$. Thus chain i and $i-1$ share their ith, $i+2$th,...$\sqrt{N}-i$th nodes. Figure 3 shows the network for $\sqrt{N}=8$.

We have $\sqrt{N}$ packets starting in a queue at the first node of each chain. Each packet simply traverses its chain to the the end. We assume that any queue has unlimited size.

Note that the packets in chain i have higher priority than those in the chain $i-1$ whenever they meet since the chain i packets travel one farther than those in chain $i-1$.

We claim that at every meeting point between chain $i-1$ and i, the packets in chain $i-1$ are delayed by all the packets in chain i. This implies the theorem since the packets in the first chain would be delayed by $\sqrt{N}$ packets at each of $\sqrt{N}/2$ meeting points, resulting in a total delay of $\Omega(N)$.

We prove the claim by induction on chain number and the number of meeting points. It certainly holds for the last two chains, i.e., the nodes of the last chain arrive at the single meeting point at the same time as those of the second to last and have higher priority. So we assume that it is true for the chain i and wish to prove that it is true for the chain $i-1$.

At the first meeting point of the chains the packets arrive at the same time since chain i has not met any other chain and the packets in chain $i-1$ are not delayed by any packet to the left. Thus the packets in chain $i-1$ are delayed. To finish, we assume that the packets in chain $i-1$ have been delayed for the first j meeting points and claim that the chain $i-1$ packets meet the chain i packets at the $j+1$st meeting point, since chain $i-1$'s packets have been delayed in the intervening node. ∎

Observation 14 *For the strategy of assigning each packet a random rank and giving priority to the packet with the lowest rank, there is an N-node network with diameter $O(\log N/\log\log N)$ and a set of paths with maximum distance $d=O(\log N/\log\log N)$ and congestion $c=O(\log N/\log\log N)$ where the expected length of the schedule is $\Omega((\log N/\log\log N)^{3/2})$.*

Proof: In this example we assume that queues have unlimited capacity. We also assume that each node can only send a message on a single output edge, without loss of generality.

The network again consists of many copies of a subnetwork.

We construct our subnetwork so that $d=c=k/\log k$. The subnetwork consists of a linear chain of length d, with loops of length $\sqrt{d}$ between adjacent nodes (see figure 4). Of the d packets, the highest priority $\sqrt{d}$ use the first $\sqrt{d}$ loops as their path. The next highest priority $\sqrt{d}$ packets use the linear chain for $\sqrt{d}$ steps and then use $\sqrt{d}-1$ loops as their path, and so on.

It is easily seen that the ith group of $\sqrt{d}$ packets delays the packets with lower priority by $d-i\sqrt{d}$ steps. Thus the last packet experiences an $\Omega(d\sqrt{d})=O((k/\log k)^{3/2})$ delay.

Once again we need the packets to be in some specific order, which can be shown to happen with high probability given enough copies of the subnetwork. As in Observation 11, it is not hard to show this requires $k=\Theta(\log N)$. ∎

6 Remarks

The scheduling algorithm from Section 3 can be used as a subroutine in algorithms for sorting and emulating shared memory machines on bounded degree networks. By using the algorithm in place of the routing algorithm in [21], it is possible to sort N packets in $O(\log N)$ steps on an N-node butterfly using constant size queues. (This observation has been made previously by Pippenger [17], Ranade [19], and Reif [20].) A shared memory machine with a large address space can be emulated by randomly hashing the memory locations to the nodes of a butterfly as in [6] and [18]. The hashing ensures that the congestion of the packets implementing each memory access step is small. The algorithm from Section 3 can be used to schedule the the movements of these packets. A more complete description of these applications will be provided in the full paper.

Acknowledgments

Thanks to Jon Greene, Johan Hastad, Charles Leiserson, Nick Pippenger, and Abhiram Ranade for helpful discussions. Thanks to Tom Cormen for producing the figures.

References

[1] M. Ajtai, J. Komlos, and E. Szemeredi, "An $O(N \log N)$ sorting network," *Proceedings of the 15th Annual ACM Symposium on the Theory of Computing*, 1983, pp. 1–9.

[2] R. Aleliunas, "Randomized parallel communication," *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1982, pp. 60–72.

[3] K. Batcher, "Sorting networks and their applications," *Proc. AFIPS Spring Joint Comput. Conf.*, 1968, Vol. 32, pp. 307-314.

[4] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, Vol. 18., 1979, pp. 143–154.

[5] R. I. Greenberg and C. E. Leiserson, "Randomized routing on fat-trees," *Advances in Computing Research*, Vol. 5, *Randomness and Computation*, S. Micali, ed., JAI Press, Greenwich, CT, 1988, to appear.

[6] A. R. Karlin and E. Upfal, "Parallel hashing — an efficient implementation of shared memory," *Proceedings of the 18th Annual ACM Symposium on the Theory of Computing*, May 1986, pp. 160–168.

[7] D. Krizanc, S. Rajasekaran, and Th. Tsantilis, "Optimal routing algorithms for mesh-connected processor arrays," *VLSI Algorithms and Architectures (AWOC 88)*, J. Reif, ed., Lecture Notes in Computer Science 319, 1988, pp. 411–422.

[8] M. Kunde, "Routing and sorting on mesh-connected arrays", *VLSI Algorithms and Architectures (AWOC 88)*, J. Reif, ed., Lecture Notes in Computer Science 319, 1988, pp. 423–433.

[9] F. T. Leighton and F. Makedon and I. Tollis, "A $2N - 2$ step algorithm for routing in an $N \times N$ mesh," unpublished manuscript.

[10] F. T. Leighton, *Complexity Issues in VLSI*, MIT Press, Cambridge, MA, 1983.

[11] F. T. Leighton, "Tight bounds on the complexity of parallel sorting," *IEEE Transactions on Computers*, Vol. C-34, No. 4, April 1985, pp. 344–354.

[12] T. Leighton and S. Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms," *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, IEEE, 1988, to appear.

[13] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*. Vol. C-34, No. 10, October 1985, pp. 892-901.

[14] C. E. Leiserson and B. M. Maggs, "Communication-efficient parallel graph algorithms for distributed random-access machines," *Algorithmica*, Vol. 3, pp. 53-77, 1988.

[15] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout, "Meshes with reconfigurable buses," *Advanced Research in VLSI: Proceedings of the Fifth MIT Conference*, J. Allen and F. T. Leighton, ed., MIT Press, Cambridge, MA, 1988, pp. 163-178.

[16] J. K. Park, "A deterministic routing algorithm for the butterfly-fat-tree," unpublished manuscript.

[17] N. Pippenger, "Parallel communication with limited buffers," *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, IEEE, 1984, pp. 127–136.

[18] A. G. Ranade, "How to emulate shared memory," *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, IEEE, October 1987, pp. 185–194.

[19] A. G. Ranade, personal communication.

[20] J. H. Reif, personal communication.

[21] J. H. Reif and L. G. Valiant, "A Logarithmic time sort for linear size networks," *Journal of the Association for Computing Machinery*, Vol. 34, No. 1, January 1987, pp. 60–76.

[22] J. Spencer, *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, PA, 1987.

[23] C. D. Thompson, *A Complexity Theory for VLSI*, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA. 1980.

[24] E. Upfal, "Efficient schemes for parallel communication," *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1982, pp. 55–59.

[25] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing*, May 1981, pp. 263–277.